

On Design of Programs
with the Accent on Programs
for Spectral Analysis

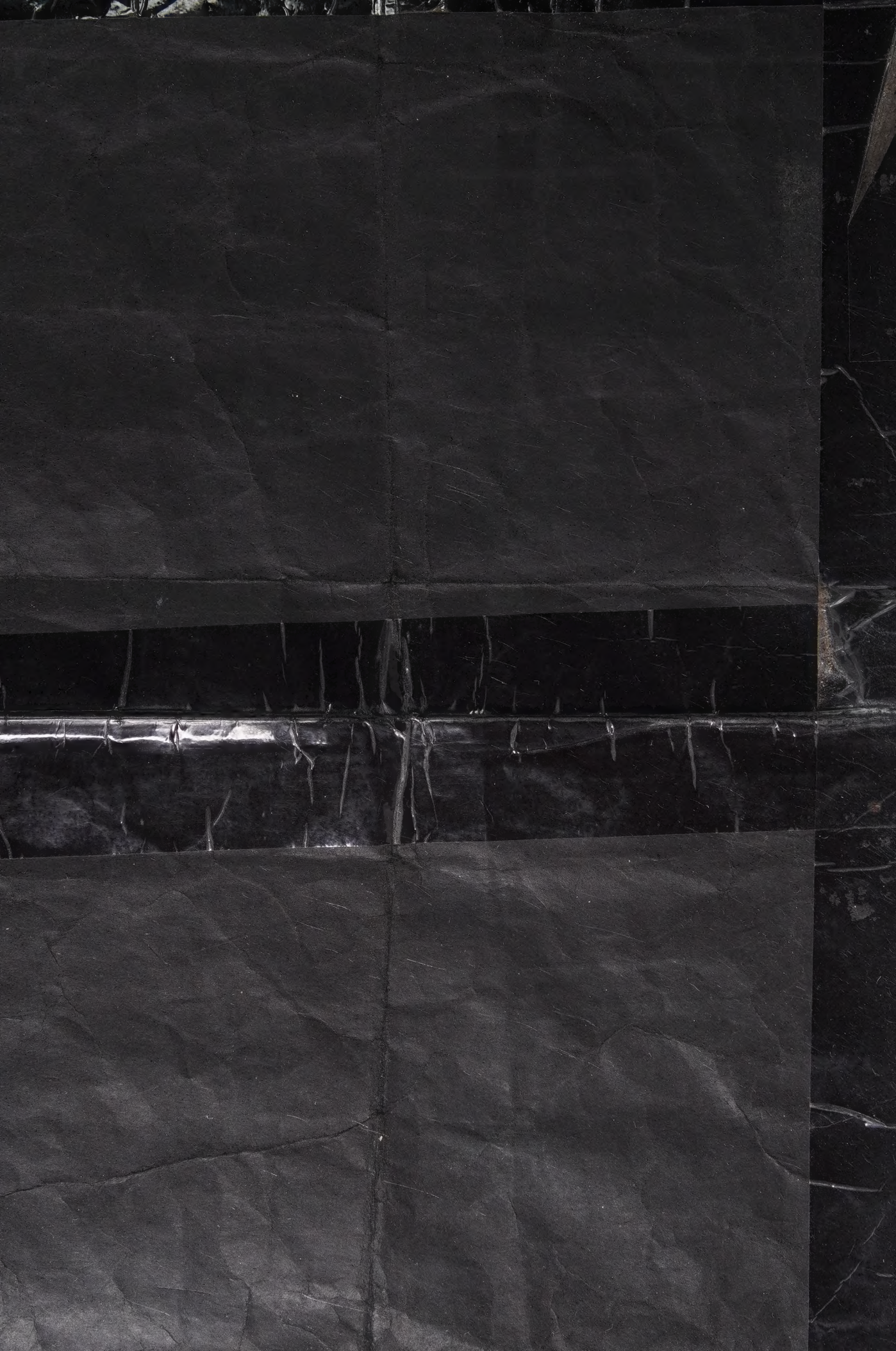
Bertil Ekenberg

CODEN: LUNFD6/(NFIP-3003)/1-38/(1978)



LUND UNIVERSITY
Department of Computer Sciences

1978



Lund University

January 1978

Department of Computer Sciences

Sölvegatan 14A

S-223 62 Lund

Sweden

On Design of Programs
with the Accent on Programs
for Spectral Analysis

Bertil Ekenberg

CODEN: LUNFD6/(NFIP-3003)/1-38/(1978)

Abstract

This report discusses problems on design and implementation of computer programs presenting graphical results for example from optimizations. The report discusses especially problems in programs for spectral analysis, such as choice of model, optimization routines, and input and output forms and devices.

Keywords and phrases: Curve fitting, spectral analysis, computer program, models, optimization methods, man-computer interaction, computer graphics, input and output devices.

Contents

1. Introduction	1
2. Characteristics of a good program	3
3. The structure of a program for spectral analysis	5
4. Choice of model	6
4.1. Choice of function	6
4.2. Implementation of functions	9
4.3. Constraints	11
4.4. Norms	11
5. Start values of parameters	13
6. Optimization methods	16
6.1. Step lengths	16
6.2. Search directions	17
6.3. Derivatives	17
6.4. Constraints	18
7. Error estimations	21
8. Input and output	22
8.1. Should a program be interactive?	22
8.2. Input considerations	24
8.3. Output devices and forms	26
9. Parts of a program for spectral analysis	30
9.1. Representation of parameters	30
9.2. The number of routines	31
9.3. Some time-saving methods	32
10. Summary	34
Acknowledgements	35
References	36

1. Introduction

The problem discussed in this report may be regarded as a curve fitting problem or a parameter estimation problem: Measurements, for example from spectral analysis, may be presented as a curve, as in Fig. 1.

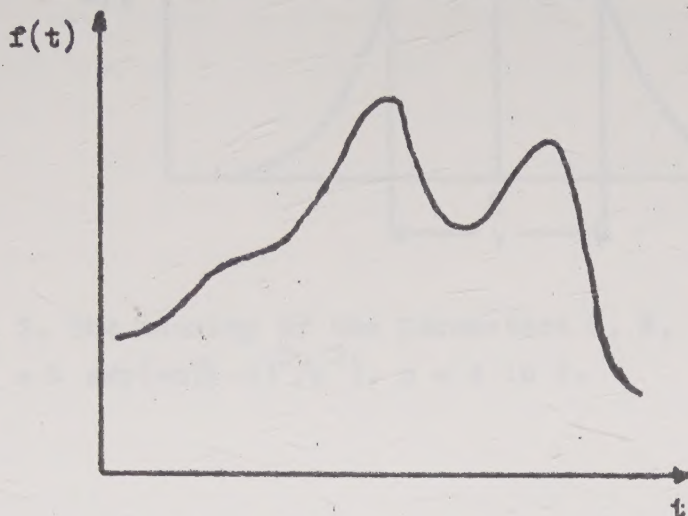


Fig. 1. Example of measurements from spectral analysis, presented as a curve.

If disturbances are disregarded, the curve is supposed to be a function of some parameters, for example

$$(1) \quad f(t) = \sum_{k=1}^n \alpha_k \exp(-c(\beta_k - t)^2 / \gamma_k^2)$$

The problem is to determine the number of components (n) and the parameters α_k , β_k , and γ_k .

If the function has the form

$$(2) \quad f(t) = \alpha \exp(-c(\beta - t)^2 / \gamma^2)$$

the parameter α corresponds to the height of the peak, β to the position, and γ to the width of the peak. With the choice $c = 4 \ln 2$, γ is equal to the width of the peak when $f(t) = \alpha/2$, as illustrated in Fig. 2.

This report discusses considerations in formulating the problem, such as choice of model, choice of optimization routines, form of input and output of the program and choice of input and output devices, and problems in implementing a program for spectral analysis for a computer. The discussion is mainly based on experience from the design and the use of a computer program described in Ekenberg [9] and [10].

The problem discussed in this report may be regarded as a finite difference problem or a parameter estimation problem (Gauss-Markov), for example from spectral analysis, may be presented as a curve, as in Fig. 1.



Digitized by the Internet Archive
in 2026

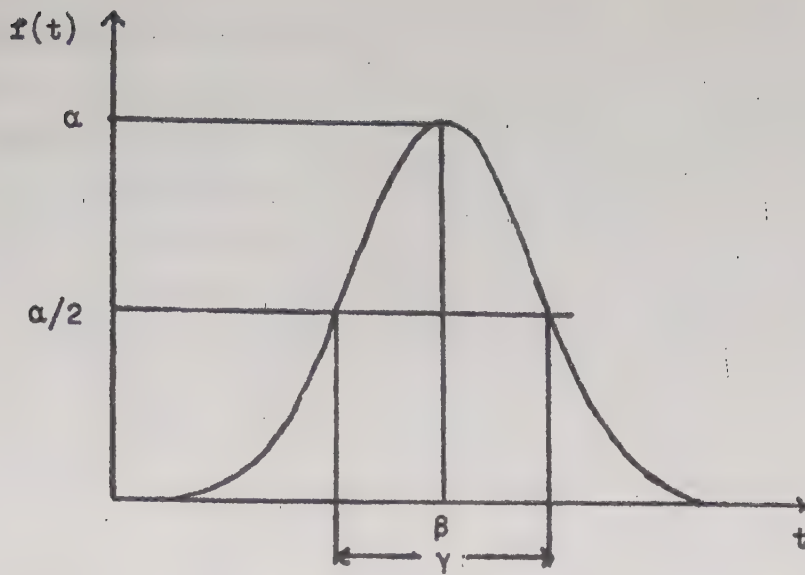


Fig. 2. The meaning of the parameters α , β , and γ when $f(t) = \alpha \exp(-c(\beta-t)^2/\gamma^2)$, $c = 4 \ln 2$.

2. Characteristics of a good program

There are many aspects on the properties of a good program. The properties may be divided into properties good for the non-programming user, the programming user, and the computer.

The non-programming user is in this report a person who wants a program that reads the measurements and presents the result with as little assistance as possible from the user. Such a user probably prefers a program for just one purpose (in other case the user would have to use control cards or something like that to tell the program what it should do). The number of parameters supplied by the user should be as small as possible. The user should not have to determine the accuracy or number of iterations of the numerical methods or the amount of printing from the program. The program should be easy to transfer from one computer to another. This means that the program should be written in a standard form of a common programming language.

The programming user is a person who is interested in using a program and in knowing how the program works and is prepared to change the program to make it suit his applications. The user may be interested in changing the model, for example the function form to a function more fitted to his problem, the numerical routines, for example to other numerical routines or to different accuracy or number of iterations, or the amount of output from the program. If some computations are unnecessary, inconvenient, incomplete or missing from the users point of view he may delete, change, or add parts of the program. To such a user it is important that the program is well documented and well structured. The non-programming user just needs to know how to use the program, but the programming user must also know how the parts of the program work and interact, and a paper describing the program and its parts is desirable as well as guiding comments in the program. If the program is changed the documentation should be changed at the same time. Machine independency is less important to the programming user than to the non-programming user since the programming user may change the program so that it makes use of special input and output devices, for example graphic displays, or other special installation dependent devices. However the program should still not be too machine dependent, for example it should not use bit manipulation for changing numbers and characters to make the program run faster on a particular computer.

The program should also use the computer in a sensible way. This means for example that the program should be fast and need little memory space. If secondary storage is needed, the program should use a suitable storage device and a suitable addressing technique, and it should avoid sending large quantities of data to slow output devices.

Although this list of properties of good programs is not complete, it shows that several demands may be made on a program, and some of the demands are in conflict with each other while some demands co-operate. Some users may find a program inconvenient if they have to specify what the program should do while other users may dislike a program if they cannot tell it what routines to use, what input and output units to use etc. A program with possibility of using different numerical routines, input and output units etc. becomes more memory-consuming and possibly slower than a program without such alternatives. If a program demands too much memory, overlay or paging techniques may have to be used to make the program run - if it is possible to run the program at all on a small computer. A program can sometimes become faster and/or less memory-consuming if the program or parts of the program is written in Assembler or other machine dependent programming languages, but this makes the program more difficult to transfer from one computer to another. General purpose routines may be slow or memory-consuming, but changes in the program might be carried out easier. Devices for computer graphics may differ much from one another, and general purpose routines for computer graphics - if they exist at all - might use properties of the devices in an inefficient way. On the other hand, programs that are small, fast and well designed for a special computer may often run to a low cost, which the user probably appreciates.

It seems to be difficult - not to say impossible - to write a program with all good properties one can wish for. A compromise between different wishes is necessary. Nevertheless one ought to give a program as many good properties as possible. Therefore one ought to specify what a good program should do and specify or order in importance what properties the program should have. In the following chapters some properties of a program for spectral analysis are discussed.

3. The structure of a program for spectral analysis

A program for spectral analysis may mainly have the following structure:

- read the measurements from the spectral analysis
- get a start approximation of parameters for the following optimization
- optimize the values of the parameters
- compute a measure of the error (or errors) of the optimized values
- show the results

Problems concerning the last four points are discussed in chapter 4 - 8. Chapter 4 discusses the choice of model. The model mainly depends on physical or chemical properties of the examined stuff, the used equipment and the way in which the equipment is used. Some implementation problems and numerical problems are presented. Chapter 5 describes some methods of finding start values of the parameters of the model (α_1, β_1 , and γ_1 for $i=1, \dots, n$ in formula (1) or corresponding parameters for other models). Chapter 6 presents problems in choosing optimization routines and similar problems. Implementation and numerical aspects are presented in chapters 5 and 6. Chapter 7 mentions some ways of estimating the error (or errors) of the approximation. Chapter 8 discusses the interaction between the user and the program in the computer including choice of input and output forms and devices. However, there are still several problems in designing and implementing a program for spectral analysis such as conflicts between different parts of the program which demand different representation of parameters (α_1, β_1 , and γ_1). Such problems are described in chapter 9.

4. Choice of model

A mathematical formulation of the curve fitting problem is: Given a set of measurements $y(t_i)$ (abbreviated y_i) for $i=1, 2, \dots, m$, find a function $f(t,x)$, where x is a set of parameters $x_1, x_2, \dots, x_p$, that minimizes $\Phi(f,y)$ (a measure of the difference between the measurements and the function). $f(t_i,x)$ is abbreviated f_i . The function (1) in the introduction is a function of the parameters $\alpha_1, \beta_1, \gamma_1, \dots, \alpha_n, \beta_n, \gamma_n$ and the function to be minimized may be $\Phi(f,y) = \sum_{i=1}^m (y_i - f_i)^2$ which is a function of the α -, β -, and γ -values. Sometimes the parameters may not have all possible values. Such restrictions may be formulated as constraints as $x_1 \leq 100$ or $x_2 = 1.1 x_5$.

4.1 Choice of function

Several different functions are used for different kinds of spectral analysis depending on different kinds of measurements and equipment that is used. Usually the measurements are described as some peaks and a background. A chemist working with ESCA spectra (ESCA = Electron Spectroscopy for Chemical Analysis) suggested that the peaks should be described by the function (1) in the introduction. Another chemist working with IR spectra (IR = Infrared) has suggested the form

$$(3) \quad f(t) = \sum_{k=1}^n \alpha_k / (1 + \gamma_k (\beta_k - t)^2 + \delta_k (\beta_k - t)^4 + \epsilon_k (\beta_k - t)^6)$$

which is a modification of the sum of Lorentz curves

$$(4) \quad f(t) = \sum_{k=1}^n \alpha_k / (1 + \gamma_k (\beta_k - t)^2)$$

after an idea from Vitek [36]. A Lorentz curve (shown in Fig. 3a) decreases more slowly in its tails than the corresponding Gaussian curve (2). An intermediate form may be described by a function in the sum (3), and this form may also be regarded as a truncated expansion in Taylor series of (2): $f(t) = \alpha \exp(-c(\beta-t)^2/\gamma^2) = \alpha / (1 + (c/\gamma^2)(\beta-t)^2 + (c^2/2\gamma^4)(\beta-t)^4 + (c^3/6\gamma^6)(\beta-t)^6 + \dots)$. Physicists working with Ge(Li) or NaI(Tl) detectors use a sum of Gaussian curves sometimes modified at the left or right or both tails with an exponential function. Lederer [19] and [20] describes each function as

$$(5) \quad f(t) = \begin{cases} \alpha \exp(T_1^2/2\sigma^2) \exp(T_1(t-\beta)/\sigma^2) & \text{if } t-\beta \leq -T_1 \\ \alpha \exp(-(t-\beta)^2/2\sigma^2) & \text{if } -T_1 \leq t-\beta \leq T_h \\ \alpha \exp(T_h^2/2\sigma^2) \exp(-T_h(t-\beta)/\sigma^2) & \text{if } t-\beta \geq T_h \end{cases}$$

where α is the height of the peak at the mean ($t=\beta$)
and $\sigma = (\text{full width at half maximum}) / (8 \ln 2)^{1/2}$
(a curve is shown in Fig. 3b)

If several peaks are examined at the same time, Lederer assumes that T_l and T_h are the same for all peaks. If one or both tails should not be regarded as exponential functions, the corresponding value of T_l or T_h is set to a very large number by the user. Bevington [4] describes several function forms including their statistical properties and physical meaning.

The background is often described as a straight line

$$(6) \quad f(t) = a t + b$$

or a quadratic function

$$(7) \quad f(t) = a t^2 + b t + c$$

Lederer [20] also permits a linear tail on the leading (low-energy) side. In this case (5) is changed to (see also Fig. 3c)

$$(8) \quad f(t) = \alpha \left(T_0 \left\{ 1 + (t - \beta) / (T_j + \sigma^2 / T_j) \right\} + \left\{ 1 - (T_0 \exp(T_j^2 / 2\sigma^2) / (1 + T_j^2 / \sigma^2)) \right\} \exp(T_j^2 / 2\sigma^2) \exp(T_l(t - \beta) / \sigma^2) \right) \text{ if } t - \beta \leq -T_l$$

$$\alpha \left(T_0 \left\{ 1 + (t - \beta) / (T_j + \sigma^2 / T_j) \right\} + \left\{ 1 - (T_0 \exp(T_j^2 / 2\sigma^2) / (1 + T_j^2 / \sigma^2)) \right\} \exp(-(t - \beta)^2 / 2\sigma^2) \right) \text{ if } -T_l \leq t - \beta \leq -T_j$$

$$\alpha \exp(-(t - \beta)^2 / 2\sigma^2) \text{ if } -T_j \leq t - \beta \leq T_h$$

$$\alpha \exp(T_h^2 / 2\sigma^2) \exp(-T_h(t - \beta) / \sigma^2) \text{ if } t - \beta \geq T_h$$

where σ , T_l , T_h , α , and β are defined as above,

T_0 is the height of the linear tail relative to the peak height, extrapolated to the peak mean ($t = \beta$),

T_j is the "joining point" of the linear tail, in channels below the mean,

and $T_0 > 0$ and $0 < T_j < T_l$

The program described in Ekenberg [9] and [10] permits the user to subtract different straight line segments in different intervals from the original curve but not to optimize the parameters of the line segments. Only one straight line and a function of form (1) or (3) can be optimized at the same time.

It is obvious that the choice of model should depend on the form of the examined data. It is reasonable to choose a model where the parameters correspond to physical or chemical properties of the examined stuff and where the meaning of the parameters is clear to the user of the program. Spectral analysis is not the same problem as finding any function that fits some given data properly. A model with few parameters is often more easy to understand and, besides, in the optimizations routines it is easier to find optimal parameters when few parameters are varied.

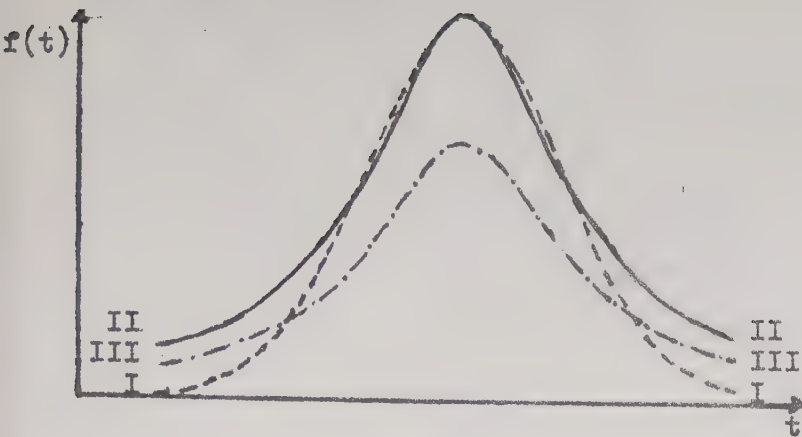


Fig. 3a

- I Gaussian curve
- II Lorentz curve
- · - · - III Lorentz curve

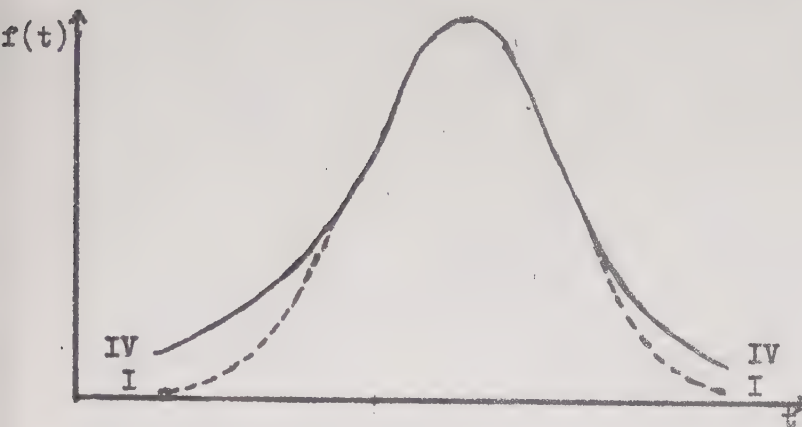


Fig. 3b

- I Gaussian curve
- IV Gaussian curve with exponential tails

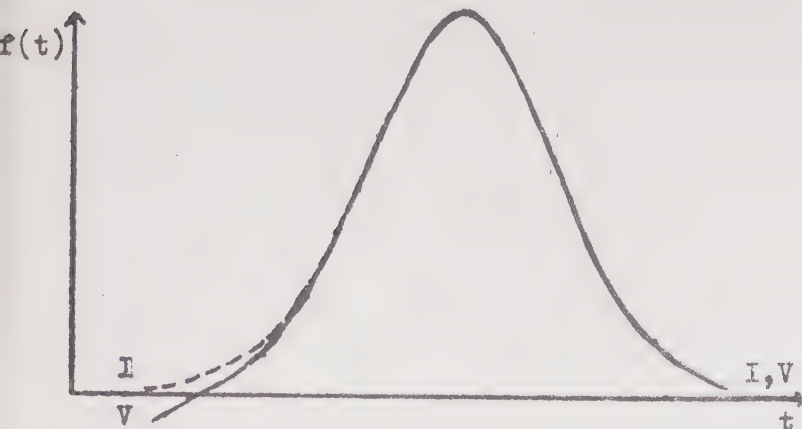


Fig. 3c

- I Gaussian curve
- V Gaussian curve with linear tail

Fig. 3. Different function forms. The first diagram contains a Gaussian curve (I) according to formula (2), a Lorentz curve (II) (formula (4)) with the same height and the same half width, and a second Lorentz curve III with the same area and the same half width as curve I. The second diagram contains the curve I and a Gaussian curve with exponential tails IV (formula (5)). The third diagram contains the curve I and a Gaussian curve with linear tail V (formula (8)).

4.2. Implementation of functions

There are several aspects on the problem of choosing and implementing functions.

One problem is: Should one or several function forms be implemented in the program? As mentioned in chapter 4.1, different applications need different function forms, mainly depending on their measurements and the equipment producing the measurements. If the user knows in advance what function form to use and does not want to change the function while the program is running, a program with only this function form is probably preferable. If different function forms are needed, but only one form each time the program is run, several versions of the program are probably to be preferred. The advantages of several versions of the program, each with one function form, are that the user does not have to tell the program what function form to use, the program does not have to test what function form to use every time the function is to be calculated, and the program needs less memory when it is run. The advantages of one program with several function forms are that the user does not have to find what version of the program to use, less secondary storage is needed for one large program than for several small programs, and changes in the program are more easily performed since the changes are only performed once. If one program includes several function forms, some complicated converting routines for the parameters may be needed, and this problem is discussed in chapter 9.

If several function forms are needed in the same program, they can be included in different ways. One way is to include a limited number of function forms and let the user choose one of these. Another way is to let the user write the explicit function form and let the program interpret or compile the text. La Fata and Rosen [16] suggest that the text should be examined and translated into reverse Polish notation. This form is later used for the function evaluations since it is less time-consuming than an examination of the text for each function evaluation. The function evaluations should become even faster if the expressions were translated into machine code and this code were executed, but this solution would be very machine dependent. The user would have to use a limited number of parameter names, operators, standard functions, and perhaps condition expressions of the form if condition then function1 else function2 and would have to supply the function form to the program every time the program is run.

In this way the user may choose function forms within very wide limits. However, this method has several disadvantages. Much programming work is needed to write an interpreter or compiler. The interpretation of the text may be time-consuming. If some kind of interpretation is needed for every function evaluation and the optimization routine (or routines) needs many function evaluations, the program may be very slow.

If an optimization routine uses partial derivatives, these may be calculated numerically, which is easy to program but may give numerical problems (discussed in chapter 6), or analytically, which demands a program for analytical derivation, and such a program demands much programming time and probably much memory and execution time.

When interpreting a function it is very difficult to take advantage of special properties of the function to make the program faster.

If the function $f(t) = a t + b$ is to be calculated for $t = t_1, t_2, \dots, t_m$ where $t_{i+1} = t_i + dt$ (constant difference), one function value can be calculated from the preceding: $f(t + dt) = f(t) + a dt$. Since the expression $a dt$ has to be calculated only once the computation of a new function value only needs one addition instead of one addition and one multiplication. If $f(t) = a \exp(b t)$ then $f(t + dt) = f(t) \exp(b dt)$ where $\exp(b dt)$ is calculated once. If $f(t) = a \exp(b t^2)$ then $f(t + dt) = f(t) \exp(2 b dt t) \exp(b dt^2)$ where $\exp(2 b dt t)$ is calculated from the preceding function value with only one multiplication as above and $\exp(b dt^2)$ is calculated only once. Such analysis may be built into a compiler but the construction of such a compiler would demand much work.

Another problem is: The function form may have to be presented to the user in one form and to optimization routines or function evaluation routines in another. For example: It may be convenient to the user to define a straight line by defining two points on the line, but routines in the program may prefer the form $a t + b$ where a and b are parameters, or perhaps even orthogonal polynomials for numerical reasons. If the function form (1) is used, Chandler [7] suggests that the parameters area, position and half-width should be used instead of height, position and half-width since the height and the half-width are highly negatively correlated. When running the program described in Ekenberg [9] and [10] it was found that the Gauss-Newton method described in [28], [33], [37], [38], and [39] converged faster when all parameters were scaled to the same size. In the same program the function (3) was implemented instead of the form suggested by Vitek [36]

$$(9) \quad f(t) = \sum_{k=1}^n \left(\alpha_k / \left(\sum_{i=0}^p (b_{ik}^2 (t - \beta_k)^2 / (i!))^{i/2} \right) \right)$$

In the form (3) but not in the form (9) a denominator may be zero or negative, but the function form should not have this property.

Other examples: If the function form is not symmetric, as for example (5) if $T_1 \neq T_h$, the mean value β and the median value m , where

$\int_{t=-\infty}^m f(t) dt = \int_{t=m}^{\infty} f(t) dt$, is not the same value, and it is not obvious whether the position of the peak should be defined as β or m .

Neither is it obvious whether the area of a single peak should be calculated from the original values or the approximation values, and in both cases whether the whole tails or only a part of the tails should be included in the area.

4.3. Constraints

When estimating parameters the user sometimes has an idea of reasonable values of the parameters or connections between the parameters. Connections between two or more peaks may be caused by physical or chemical properties of the examined material. For example, a chemist may analyse a material consisting of several components. Some of these components have known properties, that define e.g. the distance between two peaks in a composite spectrum or the area of a peak.

This knowledge should be used in the program. Sometimes a user has determined the parameters of one peak in a spectrum with several peaks and does not want these parameters to be changed later on in the optimization routines. Such properties may be included in the program as constraints, such as parameter = value, parameter $\leq$ value, parameter₁ = parameter₂ + value, parameter₁ = parameter₂ $\times$ value, parameter₁ $\times$ parameter₂ = parameter₃ $\times$ parameter₄ $\times$ value etc, as described in Ekenberg[10] and [11] and chapter 6 in this report.

4.4. Norms

A part of the model formulation is the choice of a suitable norm, i.e. a measure of the error of the approximation. This error is to be minimized.

Let y_i denote measurement number i and f_i the corresponding value of the approximating function for a given set of parameters.

A class of norms is denoted L_p -norms:

$$(10) \quad L_p = \left(\sum_{i=1}^m |f_i - y_i|^p \right)^{1/p} \quad p \geq 1$$

Norms of special interest in this case are the absolute norm

$$(11) \quad L_1 = \sum_{i=1}^m |f_i - y_i|$$

the Euclidian norm (used in least-squares approximation)

$$(12) \quad L_2 = (\sum_{i=1}^m |f_i - y_i|^2)^{1/2}$$

and the maximum norm (minimax norm, uniform norm, Chebyshev norm)

$$(13) \quad L_\infty = \max_i |f_i - y_i|$$

These norms are carefully examined in Rice [29].

Sometimes variants of the norms may be preferred. Since the norms are ≥ 0 , L_2 and L_2^2 have the same minimum and maximum points, are increasing and decreasing in the same intervals etc., so that L_2^2 may often be used instead of L_2 thus saving square root computations and often giving simplified algorithms in the optimization routines. If a value of an error using the norm is to be presented to the user, this value and the quantity used by optimization routines need not necessarily be the same. For example, L_2^2 may be minimized by an optimization routine and the value of L_2 or L_2^2/m or $(L_2^2/m)^{1/2}$ may be presented to the user (the two last forms may be preferred if the number of measurements (m) changes from time to time for example if the range of the interval of the optimization is changed and the user wants a value independent of such changes).

Another expression to be minimized (although not a norm in strict mathematical sense) is

$$(14) \quad \chi^2 = \sum_{i=1}^m ((f_i - y_i)^2 / y_i)$$

which is used for example in the program Sampo described in [31] and [32]

The expression may be regarded as a weighted sum of squares.

The choice of weights ($1/y_i$) used in nuclear physics is explained

by Bevington [4]. For physical and statistical reasons the measured

values y_i may be regarded as observations from Poisson distributions,

and have as such an estimated variance = y_i . If the errors of the

measurements are assumed to be large it is suitable to give the

corresponding terms of the sum of squares a small weight and vice versa.

The choice of weights ($1/y_i \approx 1/\text{variance}$) means that the χ^2 distribution

is used and its known statistical properties may be used in the program

(for example the χ^2 test).

5. Start values of parameters

The problem discussed in this chapter is how to give start values of the parameters to be determined by the program, such as α_k , β_k , and γ_k and n in (1). Usually optimization methods demand reasonable start values of α_k , β_k , and γ_k and take the value of n for granted.

One possibility to get start values is to ask the user to supply them. It is reasonable to have this possibility included in the program even if automatic methods may be formulated and implemented in the program since the user may know chemical or physical properties of the examined material and determine reasonable values from such properties. If the program is interactive, the user should have the possibility to change the parameters also after an optimization and use the changed parameters as start values for a second optimization. Different ways to implement this property are described in chapter 8.

However, sometimes numerical methods may be used to find start values. The problem of finding start values depends on the kind of spectra that is analysed. In chemistry, e.g. ESCA and IR spectra are analysed, and the main problem is that peaks are often composed of several components and both the determination of the number of components and the parameter values of the components of a composite peak are difficult. Physicists analysing Ge(Li) or NaI(Tl) detector spectra have often a very great number of peaks in each spectrum but each peak has usually only one component or sometimes two components but very seldom more than two components. The problem is to find relevant peaks excluding "peaks" due to random errors of the background noise and to find the number of components of the relevant peaks, while the determination of the parameter values of the peaks with only one component is a smaller problem.

Several different methods for determining start values exist. A program by Ekenberg [9], originally written for ESCA spectra but useful also for other kinds of spectra, guesses one component each time the user at the graphic display unit writes a certain command. First of all the curve describing the difference between the original measurements and the approximating curve is computed. This curve is assumed to have the form (1). After that its derivative $df(t)/dt$ and the function $g(t) = (df(t)/dt)/f(t)$ are computed numerically. If $f(t)$ is mainly one Gaussian curve in an interval, the function $g(t)$ is almost linear. For this reason $g(t)$ is approximated to a straight line in intervals around each of three points: the maximum point of $f(t)$, the maximum point of $df(t)/dt$, and the minimum point of $df(t)/dt$. The fit at the maximum point of $f(t)$

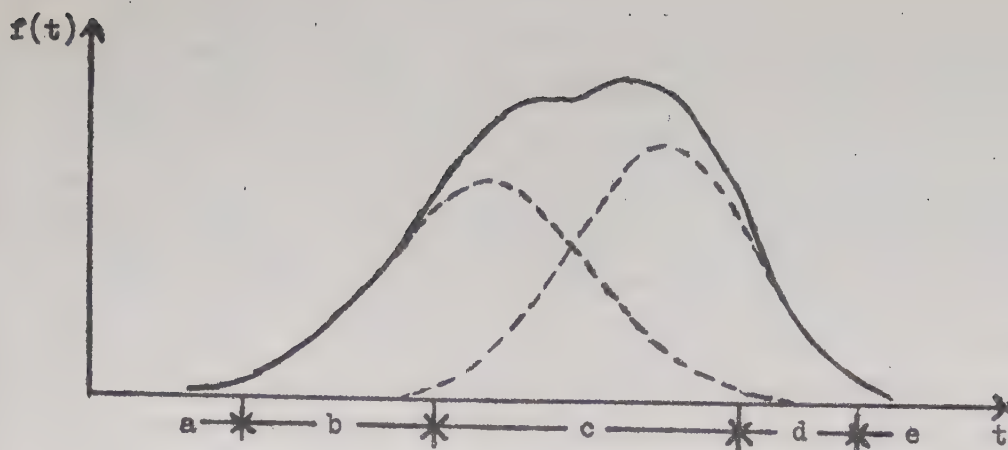


Fig. 4. A peak composed of two Gaussian curves ($n=2$ in formula (1)). Each Gaussian curve and the sum of the curves is presented. The tails of the curve (intervals a and e) are bad for fitting to one Gaussian curve since background noise dominates in measured values. In the intervals b and d the composite curve consists mainly of one Gaussian curve and good parameter estimations may be found in these intervals. Interval c is bad for fitting to one Gaussian curve since two Gaussian curves both give considerable contributions.

is probably good if $f(t)$ only consists of one Gaussian curve but bad if it consists of at least two adjacent Gaussian curves. In this case one of the other examined points may give a good result. The reason is further described in Fig. 4. The best of these three approximations of $g(t)$ to a straight line is used to determine start values of β and γ . If the interval around the maximum value of $f(t)$ was used, α is approximated to this maximum value. In other case α is set to the value of $f(t)$ when $df(t)/dt$ has its minimum or maximum (depending on the used point) multiplied by $\exp(-0.5)$ (since a function of the form (2) has the value $\alpha \exp(0.5)$ in its inflexion points). A variant of this method has also been suggested: $\log(f(t))$ is fitted to a quadratic function in an interval, and from this fit start values of α , β , and γ are computed. Another method is to fit other intervals of $g(t)$ to a straight line, for example by presenting $g(t)$ at a graphic display screen and give the user an opportunity to choose a convenient interval. Still another method is used in a program by Mukoyama [23]: $\log((f(t-1)/f(t+1))) = c(\beta-t)/\gamma^2$ (if $f(t)$ has the form (2)) is fitted to a straight line thus determining β and γ . α is determined from a weighted sum of terms $\exp(\log(y_i) + c(t-\beta)^2/\gamma^2)$, where y_i are the measured values.

In another program, Annalis by Ekström, Mauritzson, Tillman [13], an automatic analysis of Ge(Li) spectra is made. In the peak location routine γ is assumed to be linearly dependent of t , and values $c_1 = c(t_1)$, where $c_1 = \sum_k (\exp(-(k-1)^2/\gamma^2) (y_k - \bar{y} - s \bar{y}^{0.5}))$ where k assumes values depending on the calculated γ value, $\bar{y}$ is the average of the corresponding measured values y_i and s a constant given by the user. c_1 assumes positive values only where there is a peak. When the peaks are located, the measurements between the areas around the peaks are used for approximating the background to a polynomial. If the width of a peak is greater than a computed value depending on β the peak is assumed to consist of two Gaussian curves, otherwise of one Gaussian curve.

The program SAMPO, described in [31] and [32], calculates $ss_1 = dd_1/sd_1$, where dd_1 is a generalized second-difference expression and sd_1 its standard deviation, in order to locate peaks. The background is assumed to be linear, and in this case dd_1 is close to zero when the background is dominating. If ss_1 is greater than a threshold value given by the user a significant peak is found, and if ss_1 is smaller than this value but greater than another threshold value a warning is printed so that the user may determine the significance of the peak. The method is described in detail by Mariscotti [22].

The program SAMPO can also compute start values of parameters. The approximating function is assumed to be a sum of curves of the form (6) and a linear or quadratic function representing the background. The user may tell the program where typical peaks are located, and from these peaks shape-calibration parameters describing the position, width, lower tailing parameter, and higher tailing parameter are computed. All peaks are assumed to have a similar form, and start parameters for the other peaks are interpolated from the shape-calibration parameters. Start values of remaining parameters (height of each peak and background parameters) are obtained from a linear least-squares routine. Another numerical routine computes the final parameter values.

According to Routti and Prussin [32], the peak finding routine in SAMPO was chosen after comparison with a routine using a smoothed curve obtained from a Fourier transformation described in Inouge and Rasmussen [16]. The program SAMPO in turn has been compared to a program SPAN by Basu and Patro [3]. In the peak finding routine the measured curve is smoothed and the smoothed first derivative throughout the spectrum is compared with a predetermined threshold. After that several peak shape tests are applied to eliminate spurious peaks.

6. Optimization methods

Optimization routines try to find the best solution in the sense that a function of one or more variable parameters as Φ in chapter 4 finds its smallest value, i.e. finds the parameter combination that minimizes the difference (in some sense) between the measured values and an approximating function. Usually a start combination of the parameters is given, a search direction is computed and a step is taken in that direction which gives a new parameter combination with a smaller Φ value. This process is repeated until some stop criterion is fulfilled. Several optimization methods are presented and discussed in several books and reports. James [17], Brown [6], Brent [5], and Lootsma [21] are just a few examples.

6.1. Step lengths

Different optimization routines use different methods to compute step lengths when a search direction is found. In a step the function is minimized or at least almost minimized along the search direction. In a program by Ekenberg [9] and [10] three optimization routines are used: the Gauss-Newton method, also described in [28], [33], [37], [38] and [39], Nelder-Mead's Simplex routine, also described in [24], and Chandler's routine Stepit, also described in [7]. The Gauss-Newton method uses Armijó's step length rule (described in [15] and [26]). At first one step length is used, and if Φ is not smaller than a computed value (depending among other things on the last Φ value and the step length) the step length is halved and the process is repeated until a minimum value of the step length is encountered and another search direction is used instead. In the Simplex routine Parkinson and Hutchinson [25] have modified the routine so that under some circumstances the step length is multiplied by 2, 3, ... as long as Φ decreases, under other circumstances by 1/2. In the routine Stepit the step length is doubled as long as Φ decreases and finally a quadratic interpolation is made. In these routines the same start step length is used every time the routine is called, and the program might be improved by using the last used step length instead if a routine is called more than once. Another choice of step lengths is suggested by Tabata and Ito [35]: The step length is multiplied by one of the factors 1.33 ($10^{1/8}$), 1.78 ($10^{1/4}$), 3.16 ($10^{1/2}$), 10, or 10^2 depending of the history of the minimization.

6.2. Search directions

Several methods are used to find search directions. Some methods do not use derivatives: for example Stepit minimizes at first one parameter at a time, and after having minimized all parameters uses the last used step lengths for determining a new search direction. Other methods use derivatives: for example the Gauss-Newton method uses a projection or a gradient vector.

6.3. Derivatives

Partial derivatives for some optimization methods can be calculated analytically or numerically. Both methods have some advantages and some disadvantages.

For numerical reasons analytical derivatives should be used. Experiments with the Gauss-Newton method in the program by Ekenberg [9] have shown that the matrix $J^T J$ (where J is the matrix with the partial derivatives df_i/dx_k , f and x defined in chapter 4) is sometimes ill-conditioned, and since numerical derivatives are less exact than analytical ones the numerical difficulties may increase if numerical derivatives are used.

The computation of analytical derivatives is usually faster than the computation of numerical derivatives, especially if the program takes common expressions into consideration. For example, all partial derivatives of $\alpha \exp(-c(\beta-t)^2/\gamma^2)$ have in common a factor $\exp(-c(\beta-t)^2/\gamma^2)$ which only has to be computed once for each α , β , γ , and t value. If this technique is combined with the technique for function evaluations calculating one value from the preceding, as mentioned in chapter 4, still more machine time may be saved. There are still more common factors in the expressions, but using this fact to get a faster program also results in more programming work and a less clear program.

An advantage of numerical derivatives compared to analytical derivatives is that new function forms may be included in the program more easily since routines for computing partial derivatives need not be changed.

If the model used contains constraints and the constraints are implemented as a penalty function changing from one time to another, analytical derivatives are probably difficult to compute while no extra programming is needed for calculating numerical derivatives.

If new function forms are entered when the program is running and the function forms are compiled or interpreted, as mentioned but not recommended in chapter 4, numerical derivatives are probably preferable.

If a routine for calculating analytical derivatives is written, it is recommended that a routine for numerical derivatives is also used while testing the routine and the derivatives from both routines are printed and compared. This is an effective way to discover errors in the routine for analytical derivatives.

When a numerical first derivative of a function $df(x)/dx_k$ is computed, the function may be regarded as a function of one parameter x_k . If this function is called $g(x)$, $dg(x)/dx$ is usually approximated by $(g(x+h)-g(x))/h$ or $(g(x+h)-g(x-h))/(2h)$. The maximum error of the calculations is the sum of rounding errors and truncation errors and these are approximately $(2e|g(x)|/|h| + |d^2g/dx^2| |h|/2)$ and $(e|g(x)|/|h| + |d^3g/dx^3| |h|^2/6)$ respectively, and these expressions are minimized for $|h| = (4e|g(x)|/|d^2g/dx^2|)^{1/2}$ and $|h| = (3e|g(x)|/|d^3g/dx^3|)^{1/3}$ respectively. e is the relative error in the computed values of g (at best 2^{-b} where b is the number of binary digits carried by the computer in use). These expressions contain derivatives which may be difficult to compute, but d^2g/dx^2 can be approximated by $(g(x+d)-2g(x)+g(x-d))/d^2$ and d^3g/dx^3 by $(g(x+2d)-2g(x+d)+2g(x-d)-g(x-2d))/(2d^3)$ where the step d should be greater than h for numerical reasons. However, since h is not known, a reasonable d value must usually just be guessed the first time, but for a special case Bard [1] has given an algorithm for computing d^2g/dx^2 . Bard also suggests that upper and lower limits should be imposed on h , e.g. $10^{-5}|g(x)| \leq |h| \leq 10^{-2}|g(x)|$. It is to be hoped that step lengths h_k should only have to be computed once for each parameter x_k even if derivatives are computed several times.

6.4. Constraints

As mentioned in chapter 4, knowledge about the analysed spectrum can be included in a program for spectral analysis as constraints. For example, a chemist may analyse a material composed of several components and some of the components have approximately known parameters, such as peak positions, half widths, distances between peaks, or areas of peaks. Another example is when the parameters of one peak in a spectrum with several peaks have been at least approximately determined and these parameters should not be changed by optimization routines later.

Several methods to implement constraints exist. Sometimes the model may be reformulated, and unconstrained optimization methods used. Instead of saying that a parameter x should be greater than or equal to 0, the parameter may be replaced by u^2 , and if $a \leq x \leq b$ where a and b are constants, the expression $(a+b)/2 + ((b-a) \sin u)/2$ may be used instead

of x , and a value of x is calculated from the final, optimized value of u . If the approximating function has the form (1), the area of each component in the sum is $\alpha_k \gamma_k (\pi/c)^{1/2}$; and constraints on the areas are non-linear constraints. If the model is formulated with the parameters α_k , β_k , and γ_k instead, these constraints become linear constraints which are easier to treat than non-linear ones.

A simple form of constraint is the form parameter=value, for example $\alpha_1=1.5$. In the spectral analysis program by Ekenberg [9] this form of constraint is implemented in three optimization routines and the way it is implemented depends on the routine. With denotations as in chapter 4 the parameters of the function to be minimized are $x_1, x_2, \dots, x_p$. In the routine Stepit by Chandler [7] an extra integer array is used with variables = 1 if the corresponding parameter is fixed to one value and 0 otherwise. The routine starts by changing one parameter at a time except for fixed parameters, and step lengths for fixed parameters are set to zero. The routine Simplex by Nelder-Mead [24] regards the p parameters as a point in R^p and starts with $p+1$ points in R^p and changes one point at a time so that the function value is decreased in each iteration. The routine has been modified: If r of the p parameters may be changed, $r+1$ start points are given in a subspace R^r of R^p where the constraints are fulfilled. All new points are computed as linear combinations of the start points and thus always belong to the same subspace. In the routine Gaussa described in [39] all parameters are supposed to change freely. When a search direction is computed the matrix with the partial derivatives is used. If the derivatives of a fixed parameter were changed to zero, numerical problems would occur since singular matrices would appear. The method of computing search direction as if all parameters were free and then change the direction by setting step lengths of fixed parameters to zero does not work very well. Instead one should use conversion routines which order the parameters so that only free variables are regarded as input variables to the routine.

Stepit also permits constraints of the form parameter<value and parameter>value, for example $\alpha_1 \geq 1.5$. Two arrays are used in the program, one with the smallest and one with the largest permitted values. The limits should not be set too close. In this case there is a risk that the routine stops at a local minimum on the boundary of the permitted area while the method might have found a better value if it had left the permitted area at first and then entered it again.

Methods have been constructed that initially permits parameters outside specified intervals, and later force the parameters to fall within the permitted area. To the function to be minimized a penalty function $p(x,t)$ is added, and this sum is minimized instead. If, for example, the constraint

$\alpha_1 \geq 1.5$ should hold, the function $p(x,t) = ((\alpha_1 - 1.5)^2 / r)$ if $\alpha_1 \geq 1.5$, 0 otherwise) is used in Ramsin [27], and the function using $|\alpha_1 - 1.5|$ instead of $(\alpha_1 - 1.5)^2$ is used in Conn [2] and [8]. In both cases the function is minimized for different r values that gradually decrease to zero. The penalty function may also include terms describing both linear and non-linear constraints and both equality and inequality constraints and is usually easy to implement in methods not using derivatives (or at least not analytical derivatives). On the other hand, numerical problems are likely to occur when penalty functions are used.

For methods using derivatives linear constraints can be added by methods described in Bard [1], Ramsin [27], and Wedin [38]. If both equality and inequality constraints are needed and a routine only for inequality constraints is available, equalities like $\alpha_1 = 1.5\alpha_2$ may be implemented as two inequalities, $\alpha_1 \leq 1.5\alpha_2$ and $\alpha_1 \geq 1.5\alpha_2$, and if the routine only permits equalities, inequalities currently at the permitted limits are used when search directions are computed.

If needed, how should constraints be implemented in a program? One possibility is to change the program for example by adding a subroutine describing the constraints every time new constraints are needed, recompile, link and load the program and run it again. However, this is laborious, especially for interactive programs. Another possibility is to write a compiler or an interpreter for the constraints. This method has about the same advantages and disadvantages as a compiler or interpreter for functions described in chapter 4. Probably the best way to implement constraints is to limit the types of constraints to linear constraints. This may mean that the model should be reformulated so that non-linear constraints are avoided. The constraints are read from the program and stored in condensed form in one- or two-dimensional arrays. The input should be checked, but not only for syntactical errors. If, for example in an interactive program, the constraint $\alpha_1 = 1.5\alpha_2$ is given at first and later the constraint $\alpha_1 = 1.7\alpha_2$ appears, this constraint should probably replace the first one instead of just being added to the list of constraints. Other inconsistencies may be more difficult to discover, such as $\alpha_1 = 1.5\alpha_2$, $\alpha_2 = 2.5\alpha_3$, and $\alpha_3 = 1.2\alpha_1$ as constraints valid at the same time. One possibility to avoid such inconsistencies is to require that a parameter may occur in at most one constraint.

7. Error estimations

Sometimes not only the parameter values corresponding to the optimized function are desired, but also a measure of the reliability of the parameters. Bard [1] gives some general measures of errors and Bevington [4] gives several measures of errors and explains when, where and why the different measures are used in connection with problems as spectral analysis problems. One such measure is implemented in the routine Stepit. It uses the error matrix to compute standard deviations and covariances between the parameters. The method is also described in Chandler [7] and Hamilton [14]. In Rice [30] and a future report by Ekenberg and Wedin [12] a measure is described, which is based on the curvature of the fitting curve.

8. Input and output

The routines that make it possible for the user to interact with the program constitute an important part of the running system. A program can be interactive, which means that the user tells the program what to do usually depending on the results from the program. A program can also be a batch program, and in this case the user gives the program all information of what the program should do from the beginning, and when the results are computed the program stops. In the first part of this chapter is discussed whether a program for spectral analysis should be interactive or not. In the following parts input and output forms and devices are discussed.

Although this chapter mainly discusses programs for spectral analysis, it does not seem unreasonable to suppose that the points of view given here - and also in some other parts of this report - may also be adopted in similar cases, especially for the task of choosing between graphs representing different combinations of functions.

8.1. Should a program be interactive?

The answer depends much on whether a co-operation between a computer and a man gives considerably better results than a computer working alone. If decisions what to do next are often needed and programmable decision rules are difficult or impossible to formulate, the program should be interactive. For example a user can sometimes see that a computed parameter value or a measured value is unreasonable and try to give a better value before calling optimization routines again, but it may be difficult to describe in advance what should be regarded as unreasonable. It may be difficult to determine in advance if a peak consists of for example two or three Gaussian curves, but the user may try both alternatives and continue with the most promising one. But if the peaks are usually easy to examine, for example consisting of only one Gaussian curve, it is probably no idea to use an interactive program, at least not for those peaks.

If the program is interactive, results are more rapidly obtained, at least if the amount of output is small, which for example Smith and Vandoni [34] point out. The user does not have to get into the problem several times. If the input is wrong, the user may enter a new input at once if the program is written properly.

If a graphic display unit with a light pen is used, the user may choose commands by pointing at a menu of commands displayed on the display screen, which may be easier than writing the commands on the keyboard. On the other hand, numerical input is better written on a keyboard.

Besides, the possibility to point at commands with a light pen is very equipment dependent, so it is difficult to transfer such programs to other computer installations.

With interactive programs it is easy to test several parameter combinations and have the results displayed almost at once. This makes it easier for the user to get an intuitive understanding of the meaning of the parameters and parameter combinations and different function forms and optimization routines (if several are included in the program).

A disadvantage with an interactive program is that it is usually more expensive to run. On large computers the operating system must permit multiprogramming and the program must be rolled out to a disk or drum memory when not used and rolled in again when a command is given. However, it may happen that one person is permitted to use a minicomputer or a part of a minicomputer alone for a long time.

If the terminal used is a slow output device, the amount of output should be kept small. If much output is needed or paper documents are needed while results are presented on a display terminal, the program may have to write the same results or draw the same curves on two devices and in that way be more expensive to run.

There are some differences between programming batch programs and interactive programs. In both cases input should be controlled and if errors are found messages should be given. But in batch programs the program is often stopped when errors have been found. In interactive programs the user should get a description of the error and be allowed to give a new correct message or change the error in the preceding message. But sometimes standard routines for reading numerical values stop the program if the text is not a value, and in this case a special routine must check the text first. Furthermore if the program is to read a value, an interactive program first writes the name of the variable and asks the user to supply a value, but a batch program first reads the value and then writes the name of the variable and the read value. In an interactive program the user should be permitted to change a command back again. For example, if constraints can be added, it should be possible to delete constraints too, and if optimization routines fail to converge to reasonable parameter values, it should be possible to start from the old parameter values again. It is also desirable that an interactive program should be able to answer questions, for example write the current parameter values or the current active constraints. It is also desirable that commands can be given to an interactive program in arbitrary order, but in this case for example optimization routines should control that necessary parameters are supplied before the optimization starts.

8.2. Input considerations

The input to a program for spectral analysis is usually the measured values, start values of parameters, and sometimes other commands telling the program what to do. In this part of the chapter the word parameters means both the parameters to be determined by the program and parameters determining accuracy of numerical routines, amount of printing etc.

Parameters can be given values in different ways:

1. by the program designer
2. by the user
3. by the program designer, but such that it can easily be changed by the user
4. computed by the program

In the first case the parameter in, for example, an accuracy test can only be changed if the program is changed, recompiled, linked and loaded.

In the second case the values may be read from punched cards in batch programs. In this case the user must determine the values, which requires that the user knows the program and the meaning of the parameters. Often special input formats are used, i.e. the numbers (and commands) must use special columns of the card, and if a number (or command) is misplaced the result will usually not be the intended. In this case the user must be very careful when punching the cards. Some programs permit free format, and in this case several numbers may be punched after each other separated by a blank or a comma, but this is not always a standard feature in programming languages and such solutions may be machine dependent. In interactive programs the user need not know exactly in advance in what order the values should be read, since the program can present the names of the parameters one at a time and ask the user to supply a value. But also in this case the user must know the meaning of the parameters and sometimes supply the same values in several runs.

In the third case the parameters may be given values as in the second case but missing or unreasonable values are replaced by standard values. But another way is to give commands of the form

< parameter name > < is given the value > < number >

such as A=1. The user only changes parameters that should have non-standard values and parameters where the program does not have any standard values. On the other hand, the user must also write the name of the parameter.

If arrays are used, subscripted parameters may be changed by for example

B(3)=5.2

or

C(1,2)=6.

Messages from the user to the computer should be short, especially the most frequent messages, but not so short that they become difficult

to understand or remember. In this case it is possible to exclude the right bracket without causing any ambiguity. If a convention is added that parameter names must not be terminated with a digit - or preferably not contain any digit at all - the left bracket may be excluded too. In this case the messages above may be written $B_3=5.2$ and $C_{1,2}=6$ respectively. The program may be written to permit all possibilities. The expression $\langle$ is given the value $\rangle$ should be written = (the assignment symbol in the programming language Fortran) or := (the assignment symbol in Algol and Simula). In the program by Ekenberg [9] the symbol : is also permitted, because the character : but not = is a lower case character on the graphic display keyboard used by the program. Since all letters and digits are lower case characters, this facilitates the command writing. The expression $\langle$ number $\rangle$ is a real or integer value for example written according to the conventions in Fortran. A disadvantage with this way to change parameter values compared to the ways above is that it is laborious to write the program. In the programming language Simula string handling is included in the language with possibilities to compare strings with arbitrary length and to convert strings to integer or real numbers. In Fortran the text editing possibilities are small. One test can only compare at most the number of characters stored in one memory word, and this number is machine dependent. If several characters are stored in one memory word it is necessary to know exactly how the computer stores the text if a special character is to be examined. If only one character is stored in each word, it is possible to write a machine independent but memory-consuming and slow program. It is also necessary to write routines that convert strings to numbers. The names of the changeable parameters can be stored in an array (in Fortran a two-dimensional array) and limits of permitted values, number of subscripts and subscript limits for arrays can be stored in corresponding arrays. To sum up, the program should read commands as strings, examine the text one character at a time to find out the kind of command given, and if a parameter is to be given a value, split the command into parameter name, index if any, and number, and call the appropriate routines. A command similar to 'change parameter value' is 'what value has parameter'. This command may just have the form $\langle$ parameter name $\rangle$ followed by index if any.

The fourth case mentioned above is that the program computes parameter values. This is sometimes the case for parameters as α_k , β_k , and γ_k in (1), as discussed in chapter 5, and parameters for step lengths in optimization routines.

8.3. Output devices and forms

The form and the amount of the output is influenced by the resources of devices, the speed of the different devices, the possibility to use the devices interactively, and the capabilities of the operating system of the computer.

The standard form of programming languages often only defines the text output possibilities to printers and secondary storage devices (magnetic tape, drum and disk storages). Graphical output and text output to other devices require special routines, which often makes the program installation and device dependent. Graphical output is often preferred to numerical output since this facilitates the judgement of the fit of the curves.

Line printers are often found at computer installations and are often fast, at least compared to typewriter terminals. Line printers have a poor resolution compared to graphical displays, ordinary plotters and colour jet plotters but somewhat better resolution than ordinary typewriter terminals. Large errors are easily discovered if the measured and the fitted curves are drawn in the same diagram, but if the fit is good, the difference curve should be drawn too, possibly using a different scale. The difference curve may be drawn in a separate diagram beside the other curves so that it is easily discovered but two diagrams beside each other make the resolution still worse. The scale factor can be computed by the program. The measured and the fitted curves should be drawn in the same scale and so that the line printer paper is well used and the difference curve for example has the same scale as the other curves if the maximum error is greater than 5 % of the maximum value of the other curves, otherwise 10 times the scale of the other curves. If the same measured curve is drawn several times corresponding to different fits, the same scale factor should be used each time if possible, so that comparisons between the fits are facilitated. If the curves are drawn with one measured value per line, as in Fig. 5, with the largest values of the curves to the right of the line, it is advisable to compute a scale factor so that the t axis (corresponding to the measurement or difference value zero or possibly the smallest measured value, if this value is not much smaller than the maximum value) is drawn about 30 % of the width of the page from its left edge and the maximum value of the measured curve about 90 % from the edge. Rescaling is needed only when the fitting curve or the difference curve would otherwise be too large for the paper.

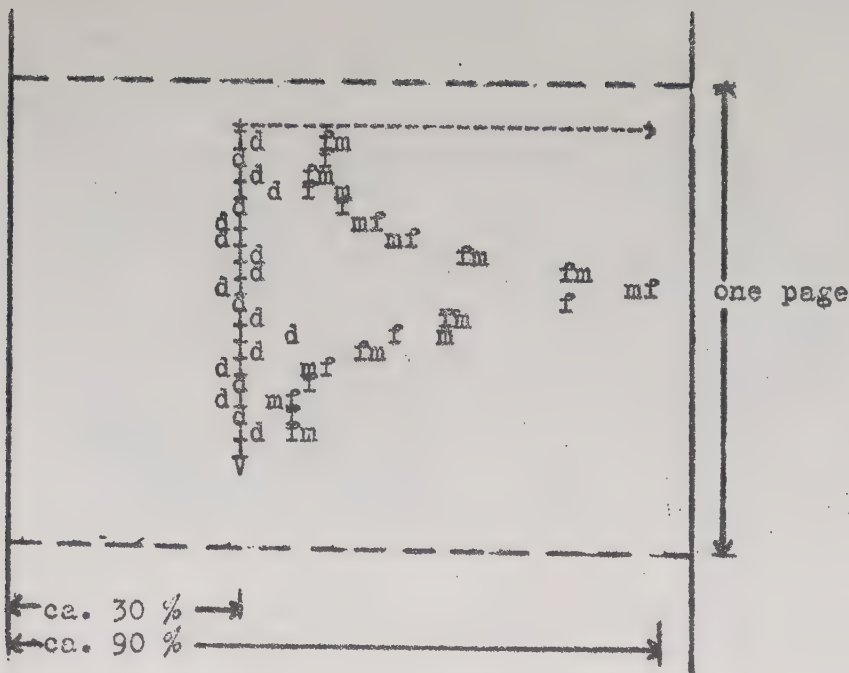


Fig. 5. A line printer page showing the measured curve (m), the fitted curve (f) and the difference between the curves ($d=m-f$). The largest measured values are to the right of the page about 90 % of the width of the page from its left edge and the values = 0 (I , if all values $\neq 0$) about 30 % from its left edge. One value per line is printed.

The different curves are printed with different characters. If several characters should have the same position on a line, it is probably most convenient just to write one of the characters. Some - but not all - line printers permit several lines to be printed without line feed, but if this possibility is used the program becomes device dependent. Diagrams can also be printed with one measurement per line position, as in Fig. 6. In this case the number of values printed is limited by the number of positions in a line and the resolution is worse if only one page is used and the program is a bit more difficult to write. If one measurement per line is written and the number of measurements exceeds the number of lines, the printing is simply continued on the next page, but with a few blank lines at the top and the bottom of each page, unless some machine dependent instructions are given to the operating system. But in both cases it is possible to limit the number of printed values by writing, for example, only every second or every third value. Whether a line printer can be used interactively or not depends mainly on the operating system. The input probably comes from a typewriter or display terminal and it is not always possible to print messages from one program on several devices at the same time. At large computer installations the output to the line printers is saved on a drum or disk storage until the program stops so that one program should not occupy a printer for a long time. But in minicomputer installations it is sometimes possible to permit the program to use all devices while the program is running.

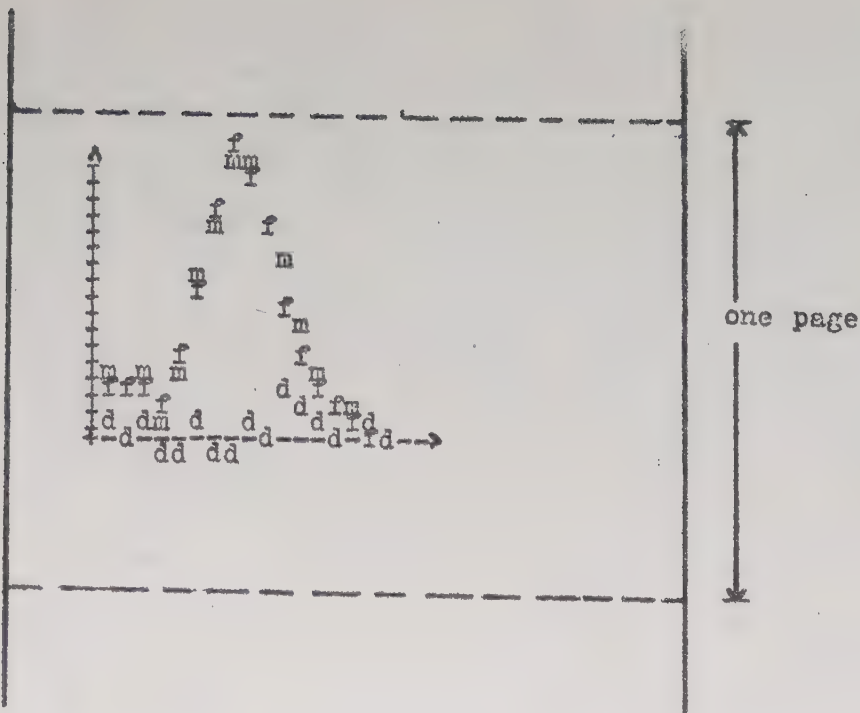


Fig. 6. A line printer page showing the measured values (m), the fitted curve (f) and the difference curve (d). One value per position on the line is printed.

Typewriter terminals are also common input and output devices, but as output device much slower than line printers. A diagram with 100 lines is printed in 10 seconds on a common line printer (600 lines/min) and ca. 3 minutes on a fast typewriter (30 characters/sec). If the program is not to be very slow the curves can only be drawn in rare cases or only ten to twenty values each time. But it might be possible to use a typewriter terminal if the curves are drawn on a fast plotter or line printer at the same time.

Alphanumeric display terminals have about the same properties as typewriter terminals. They are usually faster, but since usually only at most twentyfour lines are displayed at the same time only small diagrams can be displayed.

Graphic display terminals can draw the curves as a series of straight lines. The resolution is much better than that of alphanumeric displays, typewriters, and line printers and approximately the same as that of plotters. A Tektronix 4010 display has 1024 x 780 addressable points and a line printer page ca. 130 x 70 characters. With the same transmission speed as a fast typewriter (30 characters/sec) one curve is drawn in approximately 10 seconds and three curves in approximately 30 seconds. A diagram including axes and parameter values would take approximately one minute to produce. But this kind of terminal draws only one kind of lines, and separating the displayed curves from each other by drawing special symbols at the curves takes extra drawing and computing time.

Unlike line printers and typewriters, it also takes extra time to draw additional curves displaying the components of a composite peak.

But there are also more advanced graphic displays. In the program described by Ekenberg [9] a graphic display system Univac 1557/1558 is used. The system consists of a console, which has a cathode ray tube, a keyboard and a lightpen, and a processor with 16 K 18-bits words for display file and a part of the program, and the system is connected to a large computer Univac 1108, where more elaborate computations are performed. A great advantage with this system is the fast picture generation. If necessary data are available in the display processor, a picture is generated in a hundredth of a second, and the time it takes to change from one picture to another is neglectable to the user. It is easy to change diagrams with different fits and compare the results as the difference between the curves and the parameter values. Since the changes are easy and fast, it is possible to memorize some of the parameter values, change picture, compare with the corresponding values, remember some other values, change back and compare etc. If a Tektronix as above is used, it is a great risk that the curves or the parameter values are forgotten while the old picture is erased and the new one is drawn, so in this case it might be necessary to write down the parameters or if possible also use a typewriter terminal, a line printer, or a hard copy unit beside the Tektronix. On typewriter terminals and line printers two parameter combinations can always be written after each other so that it is easy to compare the results, but the information presented simultaneously on a graphic display is limited by the area of the display. If a large area is reserved for the diagram, the text area becomes small and vice versa. Even if the operating system of the computer permits output from one program to several devices at the same time, such programs become machine dependent. Another advantage with more advanced displays is the possibility to draw different curves with different intensities of light or for example one curve with solid lines and one with dashed lines. If the display system has its own programmable processor, commands only using this processor are performed very fast while commands using the host computer are approximately as fast as commands from typewriter terminals. A disadvantage with advanced terminals is that they are expensive and rare so that programs in general are machine dependent. And if the results are to be analysed after the run, it is necessary to write down the results or let the program present the results on another device as well.

Especially in batch programs it is possible to present curves on ordinary plotters and colour jet plotters. The different curves are marked with special symbols or drawn in different colours in order to be separated from each other. Curves drawn in different colours may also be presented on colour displays, especially in interactive programs.

9. Parts of a program for spectral analysis

It might seem that the problem of writing a program for spectral analysis is just to choose a model, program an input routine, an output routine, a routine for function evaluations, an optimization routine, and a main program calling these routines. But usually this is only a part of the problem. Several other problems appear when a program is designed and written, and some of these problems are discussed in this chapter.

9.1. Representation of parameters

One problem is how to represent the parameters that the program should determine, as α_k , β_k , γ_k , and n in (1), internally in the program. The parameters are used in different ways in different routines. If the problem is defined as in chapter 1 (formula (1)) the user may regard the parameters as one array α , one array β , and one array γ , or regard all parameters as a two-dimensional array with each column (or row) as parameters of one component (Gaussian curve) of the spectrum. So when results are printed or displayed, it seems to be natural to use a two-dimensional array. If parameters representing the background are to be added, it may be convenient to use an extra array for these parameters or redefine some of the parameters above and for example describe the function as

$$(15) \quad \text{the straight line through } (\alpha_1, \beta_1) \text{ and } (\alpha_2, \beta_2) \\ + \sum_{k=3}^n \alpha_k \exp(-(\beta_k - t)^2 / \gamma_k^2)$$

But when optimization routines are called, the parameters are often assumed to be in a one-dimensional array. The optimization routines often call a subroutine for calculating the function values $f(t_i)$ ($i=1, \dots, m$) in (1) and possibly a measure of the distance between the measured curve and the approximating curve, and this subroutine may use a one- or two-dimensional array. Sometimes the optimization routines also write results from the routines where the parameter values are written as a one-dimensional array. This makes it difficult for a user to see how the parameters are changed during the optimization even if they are written out. Besides different numerical routines often have different output formats, which further complicates comparisons between the routines. The internal parameters need not correspond directly to the parameters presented to the user. For example, it might be convenient to describe a straight line as a line passing through two points to the user and to use the form $f(t) = k t + l$ as a function of k and l in the program. As mentioned in chapter 4 and 6 the parameters α_k , β_k , and γ_k may be used as parameters instead of

α_k , β_k , and γ_k for numerical reasons. As mentioned in chapter 6, numerical problems may occur, at least when the Gauss-Newton method is used, if the parameters have considerably different magnitude, so for numerical reasons the parameters should be scaled. If constraints are used, the order of the parameters in the array may have to be changed, as mentioned in chapter 6. It may also be convenient to a user of an interactive program to define the number of components (n in (1)) as the number of $\alpha_k \neq 0$ instead of explicitly giving n a value and only use the indices $k \leq n$. In this case it is possible to guess for example three components at first and later remove the second component by just changing α_2 to 0.

One way to solve the problems is to have two arrays (or perhaps more than two arrays) to represent the parameters, a one-dimensional for the optimization routines and a two-dimensional for the input and output routines and conversion routines for transformations between these forms. If several function forms and/or several optimization routines are included in the program, several conversion routines or more complicated conversion routines are needed. If output routines are included in the optimization routines these should be changed so that the parameters are printed in user-oriented form. If possible the optimization routines and the function calculating routines should have the same parameter representation, since function calculating routines are often called several times from the optimization routines and more seldom from other parts of the program.

9.2. The number of routines

This part of the chapter discusses the problem whether one or several functions, optimization routines etc. should be included in the program.

It is sometimes reasonable to permit several function forms, especially if the user changes from one form to another when the program is run. The user may tell the program which of a given set of function forms to use by writing the number of the form and preferably have the explicit form presented from the program. If the different forms are similar to each other the model may be chosen so that these different forms are permitted. For example, a Gaussian curve with exponential tails (5) becomes a Gaussian curve if the tail parameters are fixed to very great values. But sometimes different function forms also mean that conversion routines between the forms must be programmed.

Sometimes several optimization routines may be included in the program, for example one routine may be preferred if the parameters are far from the correct ones and a different routine if parameters are almost correct. Since the error of an approximation may be measured in different ways several error routines may be preferred. Sometimes a user only wants the final results of an optimization, but sometimes several steps during the optimization should be presented in detail and several output routines preferred.

But in all these cases, using several routines also means more programming, since the program may have to include tests for example to determine what function evaluation routine to use, or more conversion routines or more complicated routines may have to be written. The program often becomes more complicated and more difficult to change. For example adding a function form does not always only mean adding some program statements including a test in the routine for function evaluations. It may also mean additions of program statements including tests in routines for calculating analytical derivatives. Input and output routines may have to be changed so that the user can choose between different function forms and have the currently used function form presented for example at a terminal. Additions in conversion routines may also be needed. Addition of a command to choose function form also means that a suitable form of the command should be formulated and implemented, so that not only a variable describing the function form is changed, but also so that the form of the command is checked and an appropriate message is given if not correct. If the number of parameters describing one component is changed, checks for reasonable indices may have to be changed, arrays redefined, different output routines programmed and used depending on the function form etc.

If several routines are included in a program, this will imply an increase of the design and the programming work, and thus the development cost. But the program may also be slower and more memory-consuming and thereby more expensive to run. If the program requires more memory than available, more memory may have to be bought, or overlay or paging techniques must be used. And some, but not all, users may find the program more complicated than necessary, as discussed in chapter 2.

9.3. Some time-saving methods

Sometimes different methods may be used in order to save time. As mentioned in chapter 6, parameters may be fixed and thus optimization routines optimize fewer parameters than used and thus becomes faster. Another method is to subtract already fitted components thus saving time both

in optimization routines and in function calculating routines. Such savings are also made if the fitted interval is as small as possible, at least if background parameters are already determined and not changed when the components of a peak are examined. The user may save time if parameters and possibly other results from optimization routines or parameters supplied by the user are saved automatically or on command from the user and recalled when needed. The user may easily compare different results and restart from old results. Results stored on secondary storages may also be recalled by the program if the program is interrupted or possibly by another program producing plots. The latter case may occur if plots only may be produced by batch programs or if plots only are produced after a manual control of the result presented on another device. Sometimes the user may want to examine several curves one after another, and in this case it is convenient to have a command to call the routine for reading measurements so that the user does not have to start the program and give start parameters etc. again when one curve is examined.

On the other hand, adding these kinds of routines may also result in disadvantages similar to those discussed in chapter 9.2.

10. Summary

This report discusses problems on design of programs for spectral analysis. There are different kinds of spectral analysis with different problems and different models, and it is probably preferable to have different programs for the different kinds of spectral analysis and not design one single program to solve all problems. Even if the same problem is to be solved at two computer installations, different programs may be needed due to different input, output or secondary storage devices or different operating systems. But there are also some common properties of spectral programs.

Measurements from spectral analysis may be presented as a curve. This curve should be approximated by a curve described by some parameters such as height, width and position of the components of the spectrum. Different models are used in different kinds of spectral analysis, and a part of the problem is to find a suitable function form. Another problem is how to measure the difference between the curves. This difference is to be minimized, and several optimization routines exist. Some of these need partial derivatives, and in this case a routine for calculating these derivatives must be programmed. The derivatives may be calculated numerically or analytically, or preferably first calculated both ways until the analytical calculations have been controlled and then only analytically. Sometimes connections between the parameters are known and may be formulated in the program as constraints of different kinds. The way to implement the constraints and the choice of optimization routine are connected with each other.

But the design of the program also depends on whether the way to treat the measurements in the program can be determined once and for all or if the treatment is changed from time to time. In the latter case the program may be ruled by commands, preferably so that the user determines what to do next from the results presented by the program. The design of the input and output routines depends on the way the program is used, but also on available resources, especially input and output devices, and the speeds and the possibilities to draw curves and present other results on these devices. The model presented to the user may be inconvenient to optimization routines or function calculating routines, and sometimes conversion routines are needed. A few other routines are also discussed in this report.

The practical application of the principles laid down in this paper may be studied in my previous papers, Ekenberg [9], [10], and [11].

Acknowledgements

I would like to thank Prof. Carl-Erik Fröberg, Dr. Per-Åke Wedin and Bengt-Erik Bengtsson for careful reading and commenting of the manuscript and for discussions which have to a great extent influenced the contents of the report.

References

1. Y. Bard: Nonlinear Parameter Estimation. Academic Press, New York and London, 1974.
2. R. H. Bartels, A. R. Conn: Linearly Constrained Discrete l_1 Problems. Research Report CORR 76-41, Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario, Canada, 1976.
3. S. K. Basu, A. P. Patro: SPAN - a Fortran Program for Routine and Detailed Analysis of Gamma Spectra using a Small Computer. Nuclear Instruments and Methods 126 (1975), 115-123.
4. P. R. Bevington: Data Reduction and Error Analysis for the Physical Sciences. McGraw-Hill Book Company, New York, San Francisco, St. Louis, Toronto, London, Sydney, 1969.
5. R. P. Brent: Algorithms for Minimization without Derivatives. Prentice-Hall, New York, 1973.
6. K. M. Brown: Computer Oriented Methods for Fitting Tabular Data in the Linear and Nonlinear Least Squares Sense. Department of Computer Science, University of Minnesota, Minneapolis, Minnesota, 1972.
7. J. P. Chandler: Subroutine Stepit. Quantum Chemistry Program Exchange (QCPE), Dept. of Chemistry, Indiana University, Bloomington, Indiana, 1965.
8. A. R. Conn, T. Pietrzykowski: A Penalty Function Method Converging Directly to a Constrained Optimum. SIAM Journal on Numerical Analysis, Volume 14 (1977), 348-375.
9. B. Ekenberg: Spektralanalys med hjälp av bildskärm. Report LU/CS 75-3, Department of Computer Sciences, Lund University, Lund, Sweden, 1975. (In Swedish.)
10. B. Ekenberg: Results from a Program for Spectral Analysis using a Graphic Display. Report LU/CS 75-5, Department of Computer Sciences, Lund University, Lund, Sweden, 1975.
11. B. Ekenberg: Curve Fitting by Use of Graphic Display, BIT 15 (1975), 385-393.
12. B. Ekenberg, P.-Å. Wedin: On Illconditioned Parameter Estimation Problems. Report LU/CS, Department of Computer Sciences, Lund University, Lund, Sweden. (To appear.)
13. P. Ekström, I. Mauritsson, J. Tillman: Annals - a Computer Program for the Automatic Analysis of Gamma-ray Spectra. Nuclear Physics Report LUNP 7101, Department of Physics, Lund University, Lund, Sweden, 1971.

14. W. C. Hamilton: Statistics in Physical Science. Ronald Press, New York, 1964.
15. T. Heverius, P-Å Wedin: Control of search direction in an approximation of the Newton-Raphson method. Department of Computer Sciences, Lund University, Lund, Sweden, 1972.
16. T. Inouge, H. C. Rasmussen: A Computer Method for the Analysis of Complex Gamma-ray Spectra. In Trans. Am. Nucl. Soc., Ann. Meeting, 1967.
17. F. James: Function Minimization. European Organization for Nuclear Research, CERN, Geneva, Switzerland. Reprinted from the Proceedings of the 1972 CERN Computing and Data Processing School, Pertisau, Austria, 10-24 September, 1972 (CERN 72-21).
18. P. LaFata, J. B. Rosen: An Interactive Display for Approximation by Linear Programming, Comm. ACM 13 (1970), 651-659.
19. C. M. Lederer: Computer Analysis of Spectra. Lawrence Radiation Laboratory, University of California, Berkeley, California.
In J. H. Hamilton, J. C. Manthuruthil (ed.): Radioactivity in Nuclear Spectroscopy. Gordon and Breach, New York, 1972, 73-107.
20. C. M. Lederer: Peak-fitting Routines for Spectral Analysis, Lawrence Radiation Laboratory, University of California, Berkeley, California.
21. F. A. Lootsma (ed.): Numerical Methods for Non-linear Optimization. Academic Press, London and New York, 1971.
22. M. A. Mariscotti: A Method for Automatic Identification of Peaks in the Presence of Background and its Applications to Spectrum Analysis. Nuclear Instruments and Methods 50 (1967), 309-320.
23. T. Mukoyama: Fitting of Gaussian to Peaks by Non-iterative Method. Nuclear Instruments and Methods 125 (1975), 289-291.
24. J. A. Nelder, R. Mead: A Simplex Method for Function Minimization. Computer Journal 7 (1965), 308-313.
25. J. M. Parkinson, D. Hutchinson: An Investigation into the Efficiency of Variants on the Simplex Method. In F. A. Lootsma (ed.): Numerical Methods for Non-linear Optimization. Academic Press, London and New York, 1971, 115-135.
26. E. Polak: Computational Methods in Optimization. A Unified Approach. Academic Press, New York, 1971.
27. H. Ransin: Ickelineär optimering. Liber Läromedel, Lund, Sweden, 1976.
(In Swedish.)

28. H. Ramsin, P-Å Wedin: A comparison of some algorithms for the nonlinear least squares problem. Report LU/CS, Department of Computer Sciences, Lund University, Lund, Sweden, 1975.
29. J. R. Rice: The Approximation of Functions. Volume 1: Linear Theory. Addison-Wesley Publishing Company, Reading, Massachusetts - Palo Alto - London, 1964.
30. J. R. Rice: The Approximation of Functions. Volume 2: Nonlinear Multivariate Theory. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts - Menlo Park, California - London - Don Mills, Ontario, 1969.
31. J. T. Routti: Sampo, a Fortran IV Program for Computer Analysis of Gamma Spectra from Ge(Li) Detectors, and for other Spectra with Peaks. UCRL-19452 UC-34 Physics TID-4500 (55th Ed), Lawrence Radiation Laboratory, University of California, Berkeley, California, 1969.
32. J. T. Routti, S. G. Prussin: Photopeak Method for the Computer Analysis of Gamma-ray Spectra from Semiconductor Detectors. Nuclear Instruments and Methods 72 (1969), 125-142.
33. A. Ruhe, P-Å Wedin: Algorithms for separable nonlinear least squares problem. Report UMINF-47.74, Department of Computer Sciences, Umeå University, Umeå, Sweden, 1974.
34. L. B. Smith, C. E. Vandoni: Graphical Man-machine Interactive Systems for Numerical Problems: PEG, a Special Purpose System; and Gamma, a General Purpose System. CERN 70-23, Data Handling Division, Geneva, Switzerland, 1970.
35. T. Tabata, R. Ito: Effective treatment of the interpolation factor in Marquardt's nonlinear least-squares fit algorithm. Computer Journal 18 (1976), 250-251.
36. A. Vitek: New Function Describing the Profiles of IR Absorption Bands. Collection Czechoslov. Chem. Commun., Vol. 39 (1974), 365-368.
37. P-Å Wedin: The non-linear least squares problem from a numerical point of view. Department of Computer Sciences, Lund University, Lund, Sweden, 1972.
38. P-Å Wedin: On least squares problems with non-linear constraints. Report LU/CS 75-4, Department of Computer Sciences, Lund University, Lund, Sweden, 1975.
39. P-Å Wedin, L. Å. Carlsson: Gaussa. LDC Program Library, Program Abstract, Lund University Computing Centre, Lund, Sweden, 1972. (In Swedish.)

